



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

Στοιχεία Φοιτητή

Επώνυμο Φοιτητή Κουσουννής

Όνομα Φοιτητή Κωνσταντίνος

Αριθμός Μητρώου p14086

Email kwstas654321@gmail.com

Ημερομηνία
Παράδοσης 14/2/2019

Κρυπτογραφία

Εργασία Χειμερινού Εξαμήνου 2018-2019

Κάθε ομάδα είναι μέχρι 3 άτομα. Ένας από κάθε ομάδα στέλνει email στο διδάσκοντα όπου δηλώνονται τα ονόματα και τα μητρώα όσων συμμετέχουν στην ομάδα καθώς και 2 εργασίες (θα ανατεθεί μόνο μια στην κάθε ομάδα). Ημερομηνία παράδοσης: Τελευταία ημέρα της εξεταστικής του χειμερινού εξαμήνου.

Η εργασία που μου ανατέθηκε Κουσουννής Κωνσταντίνος με AM:p14086 είναι η 3.

3. Γράψτε ένα πρόγραμμα σε γλώσσα προγραμματισμού της αρέσκειας σας το οποίο θα κρυπτογραφεί μια πιστωτική κάρτα σύμφωνα με τις οδηγίες που θα βρείτε εδώ.

[https://blog.cryptographyengineering.com/2011/11/10/format-preserving-encryption-or-how-to/](https://blog.cryptographyengineering.com/2011/11/10/format-preserving-encryption-or-how-to/(p14086))
(p14086)

Περίληψη Εργασίας

Διατήρηση Μορφής Κρυπτογράφησης ή πώς να κρυπτογραφήσετε ένα αριθμό πιστωτικής κάρτας με τον AES.

Αρχικά έχουμε έναν αριθμό πιστωτικής κάρτας ο οποίος αποτελείται από 16 ψηφία .Αυτά τα 16 ψηφία θέλουμε να τα μετατρέψουμε σε δυαδική μορφή τα οποία θα αποτελούνται από 54 ψηφία 0 και 1 .

Όταν όμως μετατρέπουμε αυτά τα 16 ψηφία σε δυαδική μορφή δεν μπορούμε να χρησιμοποιήσουμε την πρότυπη μορφή κρυπτογράφησης του AES γιατί το μέγεθος μπλοκ είναι 128 bits, ενώ του DES είναι 64 bits. Αν προσπαθήσουμε να το κάνουμε κρυπτογράφηση δεν θα μπορέσουμε να γυρίσουμε πίσω στην δεκαδική μορφή του δεκαεξάρικου .

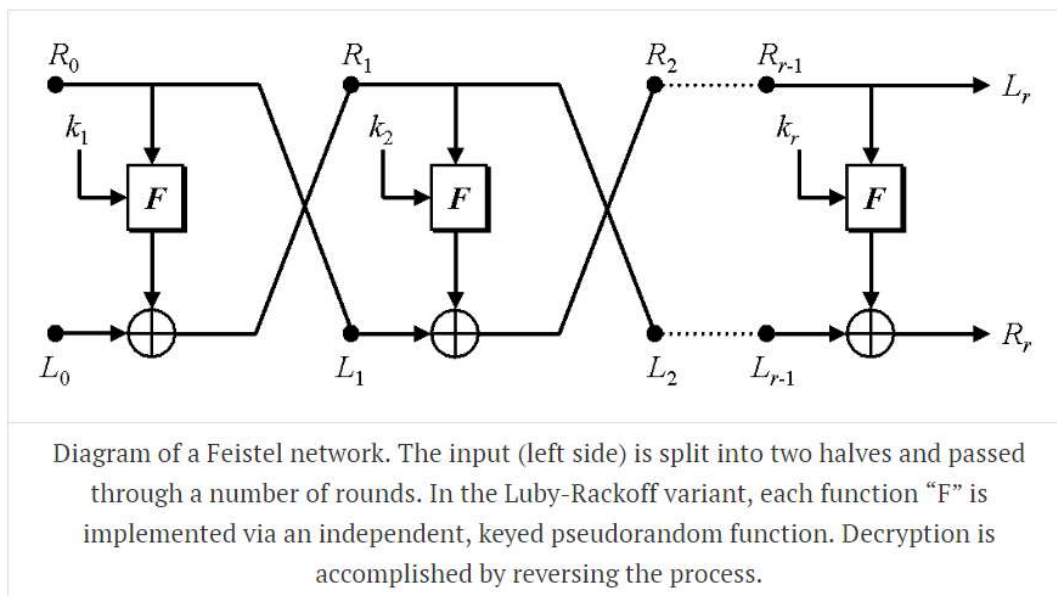
Οπότε για να μπορέσουμε να κρυπτογραφήσουμε την δυαδική μορφή θα χρησιμοποιήσουμε την **μέθοδο διατήρησης μορφής Κρυπτογράφησης**.

https://en.wikipedia.org/wiki/Format-preserving_encryption

Η κύρια ιδέα αυτής της μεθόδου είναι να χτίσουμε μια κρυπτογράφηση η οποία λειτουργεί σε ένα 54-bit μπλοκ , για παράδειγμα , μπορούμε να κόψουμε την είσοδο σε δύο μέρη των 27-bit , L0 και R0 , και να τα περάσουμε μέσα σε ένα Feistel δίκτυο για μερικούς γύρους .Απαιτεί μερικές **XOR** και μερικές **μη γραμμικές συναρτήσεις F()**,της οποίες θα ενσωματώσουμε χρησιμοποιώντας μια είδη υπάρχον κρυπτογράφηση **όπως του AES** .

Σημείωση , κάθε φορά που καλούμε τον AES πρέπει να χρησιμοποιήσουμε ένα διαφορετικό κλειδί το οποίο θα προκύπτει μέσα από ένα αρχικό κλειδί δηλαδή όλα τα υποκλειδιά που θα δημιουργούμε θα πηγάσουν από το αρχικό κλειδί μας .Αυτή την τεχνική θα την κάνουμε χρησιμοποιώντας πρότυπες τεχνικές . Μιας και θα χρησιμοποιήσουμε την Κρυπτογράφηση AES η οποία είναι καλύτερη τεχνική από

την DES S-boxes, άρα δεν χρειάζεται να τρέξουμε το Feistel δίκτυο για 16 γύρους αλλά για 3 με 4 γύροι είναι αρκετοί.



Παραπάνω μπορούμε παρατηρήσουμε το **(Feistel Network) δίκτυο Feistel**. Όπως βλέπουμε έχουμε από την μία μεριά τα **R0=27-bit** και από την άλλη μεριά έχουμε το **L0=27-bit**. Στην συνέχεια περνάμε το **R0** μαζί με το **κλειδί k** μέσα από την συνάρτηση **F()** και στην συνέχεια κάνουμε **XOR** το αποτέλεσμα που προκύπτει από την συνάρτηση με το **L0**. Στην συνέχεια αναθέτουμε το αποτέλεσμα από το XOR μας ως **R1** ενώ το **L1** παίρνει την τιμή **R0** και επαναλαμβάνουμε την διαδικασία ανάλογα με τους γύρους μας.

Αυτό δεν είναι αρκετό θα πρέπει να ακολουθήσουμε μια διαδικασία η οποία ονομάζεται **Cycling**

Cycling

Black and Rogaway είχαν μερικές ιδέες πάνω σε αυτό. Μία από αυτές τις **προσεγγίσεις** είναι κάτι που ονομάζεται **'Cycling'**

Ο τρόπος με τον οποίο δουλεύει είναι ο εξής:

Αρχικά έχουμε ένα αριθμό με 16 ψηφία, έστω ο αριθμός πιστωτικής κάρτας Visa ("4532294977918448") ακολουθώ τα εξής βήματα.

1. Αρχικά θα κωδικοποιήσω τον αριθμό σαν 54-bit ακεραίων (το μετατρέπω δηλαδή στην δυαδική του μορφή), στην συνέχεια θα κρυπτογραφήσω αυτόν τον κωδικοποιημένο αριθμό με την μέθοδο που αναφέραμε παραπάνω με το **δίκτυο του Feistel (Feistel Network)** δηλαδή θα το περάσω μέσα από ένα 54-bit block cipher.

2. Στο τέλος της διαδικασίας θα πρέπει να έχω ένα 54-bit κρυπτογραφημένο κείμενο με ονομασία **C**. Τώρα υπάρχει μια πολύ καλή πιθανότητα ότι, αναπαριστάμενο αυτό το κρυπτογραφημένο κείμενο σαν ακέραιος, το κρυπτογραφημένο κείμενο με ονομασία **C** θα είναι αρκετά μικρό ώστε να

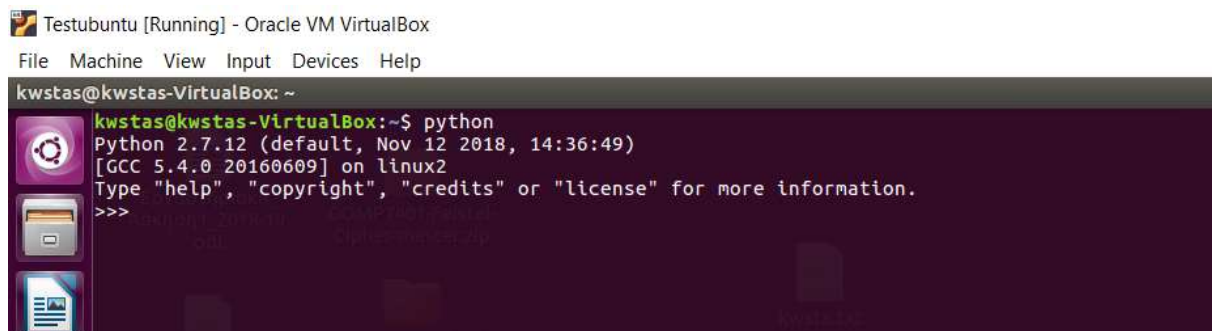
αναπαριστάτε σαν ένας ακέραιος με 16 ψηφία .Αν αυτό γίνει τότε ολοκληρώνεται η διαδικασία με επιτυχία και αυτή είναι η έξοδος μου δηλαδή το τελικό κρυπτογραφημένο αποτέλεσμα.

3.Αλλά τι γίνεται αν το κρυπτογραφημένο κείμενο C είναι πολύ μεγάλο ? Κανένα πρόβλημα .Αν διαδικασία δεν ολοκληρωθεί με επιτυχία με την πρώτη τότε ξανά επαναλαμβάνουμε την διαδικασία μας. Παίρνουμε το Κρυπτογραφημένο κείμενο που προέκυψε C ,και το ξανά κρυπτογραφούμε για να προκύψει ένα νέο C πηγαίνοντας στο **Βήμα 2**.

Υλοποίηση και Ανάλυση Εργασίας

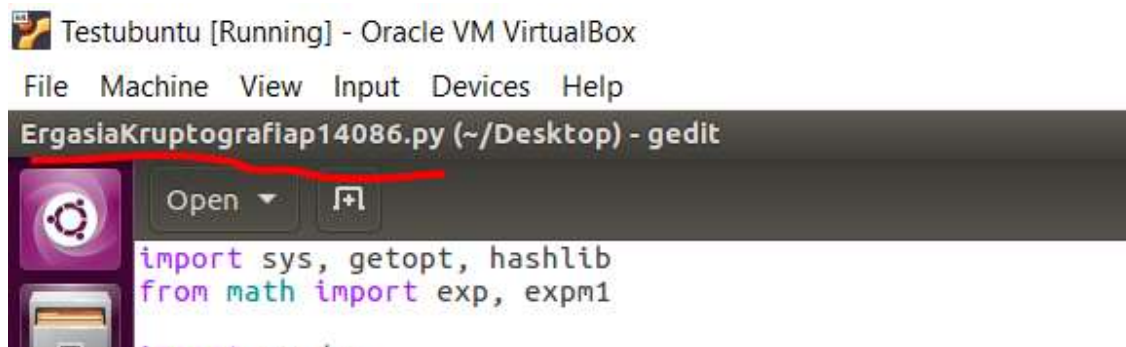
Προγραμματιστικό Εργαλείο Υλοποίησης Εργασίας

Θα υλοποιήσω το πρόγραμμα μου σε **Python 2.7.12**



```
Testubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
kwstas@kwstas-VirtualBox: ~
kwstas@kwstas-VirtualBox:~$ python
Python 2.7.12 (default, Nov 12 2018, 14:36:49)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Στην συνέχεια δημιουργούμε ένα αρχείο που θα γράψουμε τον κώδικα μας της μορφής **ErgasiaKruptografiap14086.py**



```
Testubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
ErgasiaKruptografiap14086.py (~/.Desktop) - gedit
Open [+]
import sys, getopt, hashlib
from math import exp, expm1
```

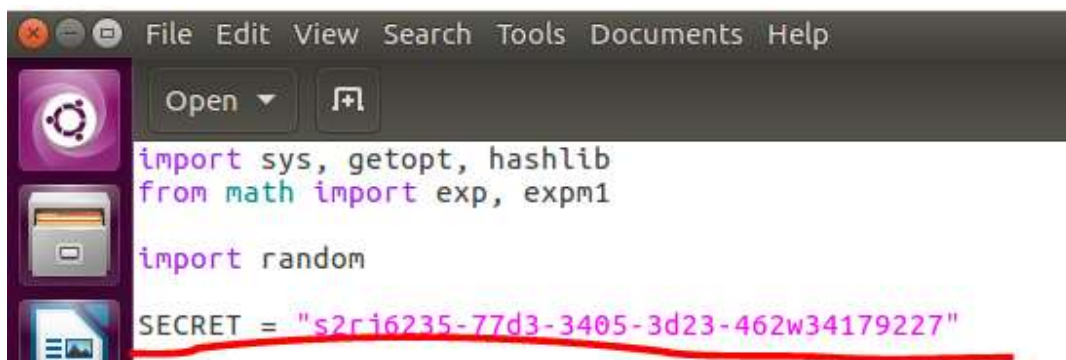
Αρχικά ενσωματώνουμε τις βιβλιοθήκες που θα χρησιμοποιήσουμε στην συνέχεια για να υπολογίσουμε τις παρακάτω μεταβλητές .



import sys, getopt, hashlib

from math import exp, expm1 //θα το χρησιμοποιήσουμε για την δημιουργία κατακερματισμένων μεταβλητών

import random //για την παραγωγή τυχαίων αριθμών



Έχω έναν secret αριθμό τον οποίο και θα χρησιμοποιήσω όταν δημιουργώ κλειδιά μου .

```
CreditCardNumber=4532294877918448 #Its an example of a credit card number that the article give us.  
key= ''.join(chr(random.randint(0,0xFF))for i in range(16)) #create a random 16 byte key.  
BinaryFormOfCreditCardNumber=bin(CreditCardNumber)[2:] #Calculate the binary form of the credit card number.  
ROUNDS = 4  
BLOCKSIZE = 54  
BLOCKSIZE_BITS = 54
```

Αρχικά δηλώνω τον αριθμό της Visa χρεωστικής κάρτας που αποτελείτε από 16 αριθμούς τον οποίο και μας προτείνει και στην εκφώνηση της εργασίας .Μπορούμε να αντικαταστήσουμε τον αριθμό με οποιοδήποτε 16 ψηφία θέλουμε.

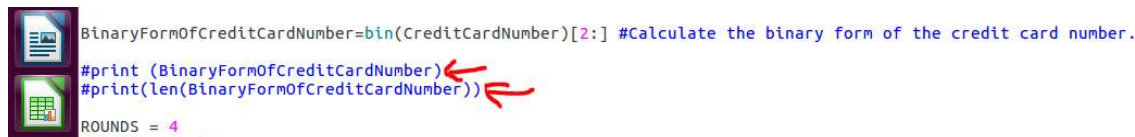
CreditCardNumber=4532294877918448

key="".join(chr(random.randint(0,0xFF)) for i in range(16))

Στην συνέχεια δημιουργώ ένα τυχαίο δεκαεξάρικο κλειδί.

BinaryFormOfCreditCardNumber=bin(CreditCardNumber)[2:]

Εδώ μετατρέπω αυτόν τον 16 ψηφίο αριθμό στην δυαδική του μορφή δηλαδή σε μορφή 0 και 1



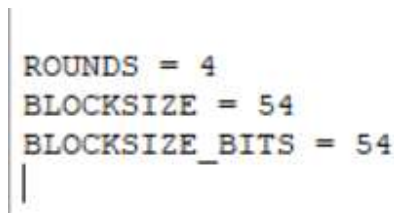
```
BinaryFormOfCreditCardNumber=bin(CreditCardNumber)[2:] #Calculate the binary form of the credit card number.
#print (BinaryFormOfCreditCardNumber)
#print(len(BinaryFormOfCreditCardNumber))
ROUNDS = 4
BLOCKSIZE = 54
```

Μπορούμε αν θέλουμε να βγάλουμε από τα σχόλια την εκτύπωση στην κονσόλα την δυαδική μορφή της χρεωστικής κάρτας και του μήκους αυτής της δυαδικής μορφής για να επιβεβαιώσουμε ότι εκτυπώνονται σωστά σε δυαδική μορφή.



```
kwstas@kwstas-VirtualBox:~/Desktop$ python ErgasiaKryptografiap14086.py
100000000110100001100100100010001101110011010011110000
53
```

Όπως μπορούμε να δούμε εμφανίζεται σε δυαδική μορφή ο αριθμός και τα ψηφία είναι 53 για αυτόν τον αριθμό.



```
ROUNDS = 4
BLOCKSIZE = 54
BLOCKSIZE_BITS = 54
```

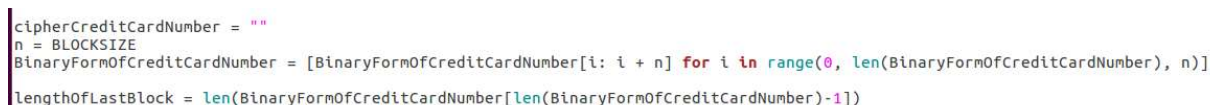
Στην συνέχεια ορίζουμε τους γύρους μας οι οποίοι θέλουμε να είναι 4

ROUNDS = 4

BLOCKSIZE = 54

Ορίζουμε το μέγεθος μπλοκ σε 54

BLOCKSIZE_BITS = 54



```
cipherCreditCardNumber = ""
n = BLOCKSIZE
BinaryFormOfCreditCardNumber = [BinaryFormOfCreditCardNumber[i: i + n] for i in range(0, len(BinaryFormOfCreditCardNumber), n)]
lengthOfLastBlock = len(BinaryFormOfCreditCardNumber[len(BinaryFormOfCreditCardNumber)-1])
```

Στην συνέχεια αρχικοποιώ την μεταβλητή που βάζω το κρυπτογραφημένο μήνυμα της χρεωστικής κάρτας.

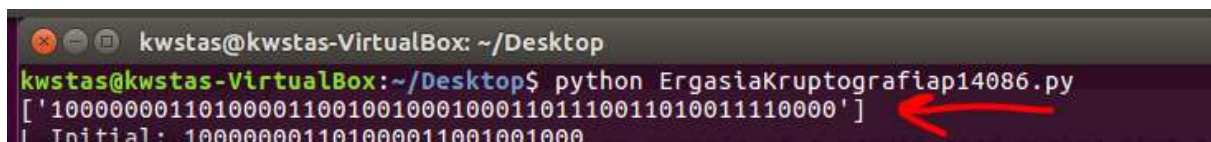

```
cipherCreditCardNumber = ""
```

```
n = BLOCKSIZE //παίρνω το μέγεθος μπλοκ
```

Στην συνέχεια εδώ βάζω αυτόν τον δυαδικό αριθμό μέσα σε αγκύλες και στη συνέχεια σε αυτάκια σαν ένας πίνακας τα δυαδικά ψηφία.

```
BinaryFormOfCreditCardNumber = [BinaryFormOfCreditCardNumber [i: i + n] for i in range (0, len  
(BinaryFormOfCreditCardNumber), n)]
```

Όπως βλέπουμε και παρακάτω.



```
kwstas@kwstas-VirtualBox: ~/Desktop
kwstas@kwstas-VirtualBox:~/Desktop$ python ErgasiaKruptografiap14086.py
['100000000110100001100100100010001101110011010011110000']
Initial: 1000000001101000011001001000
```

Εδώ επειδή θέλουμε να χωρίσουμε το δυαδικό αριθμό σε 27-bit για το R0 και L0 ορίζουμε μέγεθος μπλοκ με 54-bit που σημαίνει ότι θα χωρέσει όλος ο δυαδικός αριθμός μας.

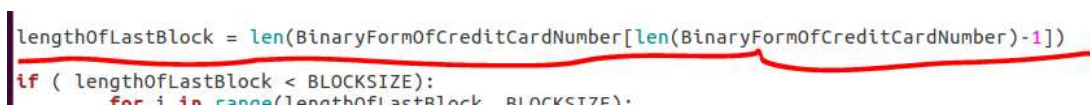
Σε περίπτωση που δηλώναμε μικρότερο μέγεθος μπλοκ για παράδειγμα 16 θα βλέπαμε ότι ο δυαδικός αριθμός μας θα χωριζόταν σε ομάδες των 16

Παράδειγμα:



```
kwstas@kwstas-VirtualBox:~/Desktop$ python ErgasiaKruptografiap14086.py
['10000000011010000', '1100100100010001', '1011100110100111', '10000']
Initial: 100000000
```

Όπως βλέπουμε έχει χωριστεί σε 16+16+16+5.



```
lengthOfLastBlock = len(BinaryFormOfCreditCardNumber[len(BinaryFormOfCreditCardNumber)-1])
if (lengthOfLastBlock < BLOCKSIZE):
    for i in range(lengthOfLastBlock, BLOCKSIZE):
```

```
lengthOfLastBlock = len (BinaryFormOfCreditCardNumber [len (BinaryFormOfCreditCardNumber)-1])
```

Παίρνουμε το μήκος μεγέθους μπλοκ .

```

if ( lengthOfLastBlock < BLOCKSIZE):
    for i in range(lengthOfLastBlock, BLOCKSIZE):
        BinaryFormOfCreditCardNumber[len(BinaryFormOfCreditCardNumber)-1] += " "

TemporaryKey = mykey(key)
MyFirstKey = TemporaryKey

for block in BinaryFormOfCreditCardNumber:
    #print ("Block: " + block)
    L = [""] * (ROUNDS + 1)
    R = [""] * (ROUNDS + 1)
    L[0] = block[0:BLOCKSIZE/2]
    R[0] = block[BLOCKSIZE/2:BLOCKSIZE]
    print ("L Initial: " + L[0])
    print ("R Initial: " + R[0])
    for i in range(1, ROUNDS+1):

        L[i] = R[i - 1]
        if (i == 1):
            Key = MyFirstKey
        else:
            Key = SubKeyGeneration(L[i], MyFirstKey, i)

        R[i] = xor(L[i - 1], My_Function(R[i - 1], i, Key))

    cipherCreditCardNumber += (L[ROUNDS] + R[ROUNDS])

```

Στην συνέχεια αν το μήκος μεγέθους μπλοκ είναι μικρότερο από το μπλοκ αρχικοποιούμε το **BinaryFormOfCreditCardNumber**

```

if (lengthOfLastBlock < BLOCKSIZE):

    for i in range (lengthOfLastBlock, BLOCKSIZE):

        BinaryFormOfCreditCardNumber [len (BinaryFormOfCreditCardNumber)-1] += " "

```

Στην συνέχεια δημιουργούμε ένα κλειδί με το τυχαίο κλειδί που φτιάξαμε παραπάνω και το ορίζουμε ως προσωρινό κλειδί γιατί αυτή η μεταβλητή θα μεταβάλλεται καθώς κάθε φορά που θα αλλάζουμε γύρο θα δημιουργούμε και ένα υποκλειδί με βάση το αρχικό κλειδί.

TemporaryKey = mykey(key)

Για να δημιουργήσουμε το πρώτο κλειδί μας καλούμε την συνάρτηση που έχουμε δημιουργήσει παραπάνω

```

def mykey(key):
    return hashlib.sha256(key + SECRET).hexdigest()

```

hashlib. sha256(key + SECRET). hexdigest ()

Καλούμε την συνάρτηση από την έτοιμη βιβλιοθήκη που κάναμε import στην αρχή και μαζί με τον προσωπικό κωδικό μας (**secret**) που ορίσαμε στην αρχή δημιουργούμε το αρχικό μας κλειδί .

```
import sys, getopt, hashlib
from math import exp, expm1
```

Στην συνέχεια αρχικοποιούμε και το πρώτο μας κλειδί το οποίο θα χρησιμοποιήσουμε για να δημιουργήσουμε όλα τα υποκλειδιά μας.

MyFirstKey = TemporaryKey

```
MyFirstKey = TemporaryKey

for block in BinaryFormOfCreditCardNumber:
    #print ("Block: " + block)
    L = [""] * (ROUNDS + 1)
    R = [""] * (ROUNDS + 1)
    L[0] = block[0:BLOCKSIZE/2]
    R[0] = block[BLOCKSIZE/2:BLOCKSIZE]
    print ("L Initial: " + L[0])
    print ("R Initial: " + R[0])
    for i in range(1, ROUNDS+1):

        L[i] = R[i - 1]
        if (i == 1):
            Key = MyFirstKey
        else:
            Key = SubKeyGeneration(L[i], MyFirstKey, i)

        R[i] = xor(L[i - 1], My_Function(R[i - 1], i, Key))

    cipherCreditCardNumber += (L[ROUNDS] + R[ROUNDS])

print(cipherCreditCardNumber)
print(len(cipherCreditCardNumber))
```

Εδώ ξεκινάμε το Feistel δίκτυο μας.

```
for block in BinaryFormOfCreditCardNumber:
    #print ("Block: " + block)
    L = [""] * (ROUNDS + 1)
    R = [""] * (ROUNDS + 1)
```

Δημιουργώ μια For επανάληψη η οποία θα επαναληφθεί με βάση όσα μπλοκ έχουμε δημιουργήσει από το προηγούμενο παράδειγμα μας τις αγκύλες που αναφερθήκαμε παραπάνω.

Για το δικό μας το παράδειγμα και σύμφωνα με τις οδηγίες μας από την εκφώνηση θέλουμε να αναθέσουμε στο L0=27-bit και R0=27-bit δηλαδή R0 θα πάρει το πρώτο μισό από τον δυαδικό μας αριθμό και το L0 το δεύτερο μισό από τον δυαδικό μας αριθμό .

Αρχικά αρχικοποιούμε τους πίνακες μας L και R με βάση το μέγεθος των γύρων μας.

Εδώ έχουμε ορίσει τον **γύρο** μας σύμφωνα με την εκφώνηση μας δηλαδή **4**. Στην εκφώνηση μας προτείνει να χρησιμοποιήσουμε 3 με 4 γύρους εγώ χρησιμοποιώ 4 γύρους όπως αρχικοποιήσαμε και παραπάνω.

Εδώ αρχικοποιώ με βάση τους γύρους μου

for block in BinaryFormOfCreditCardNumber:

```
#print ("Block: " + block)
```

```
L = [""] * (ROUNDS + 1)
```

```
R = [""] * (ROUNDS + 1)
```

Στην συνέχεια αναθέτω σε κάθε ένα από τα δυο $R0 = 27\text{-bit}(54/2)$ από τον δυαδικό αριθμό μου δηλαδή το πρώτο μισό και $27\text{-bit}(54/2)$ από τον δυαδικό αριθμό μου δηλαδή το δεύτερο μισό.

```
L[0] = block[0: BLOCKSIZE/2]
```

```
R[0] = block[BLOCKSIZE/2: BLOCKSIZE]
```

Εδώ έχω βάλει σχόλιο στα αποτελέσματα έτσι ώστε να μην εμφανίζονται στην κονσόλα μου.

```
#print ("L Initial: " + L[0])
```

```
# print ("R Initial: " + R[0])
```

Αν θέλω να επιβεβαιώσω το L[0] και R[0] αφαιρώ την δέση για να μην είναι σχόλιο και εμφανίζω στην κονσόλα το χωρισμένα μέρη.

Παράδειγμα:



```
for block in BinaryFormOfCreditCardNumber:
    #print ("Block: " + block)
    L = [""] * (ROUNDS + 1)
    R = [""] * (ROUNDS + 1)
    L[0] = block[0:BLOCKSIZE/2]
    R[0] = block[BLOCKSIZE/2:BLOCKSIZE]
    print ("L Initial: " + L[0])
    print ("R Initial: " + R[0])
    for i in range(1, ROUNDS+1):
```

Αφαιρώ την δέση το σχόλιο μου δηλαδή.

```
kwstas@kwstas-VirtualBox: ~/Desktop
kwstas@kwstas-VirtualBox:~/Desktop$ python ErgasiaKruptografiap14086.py
L Initial: 1000000001101000011001001000
R Initial: 10001101110011010011110000
```

L initial:1000000001101000011001001000

R initial:10001101110011010011110000

```
print ("L Initial: " + L[0])
print ("R Initial: " + R[0])
for i in range(1, ROUNDS+1):

    L[i] = R[i - 1]
    if (i == 1):
        Key = MyFirstKey
    else:
        Key = SubKeyGeneration(L[i], MyFirstKey, i)

    R[i] = xor(L[i - 1], My_Function(R[i - 1], i, Key))

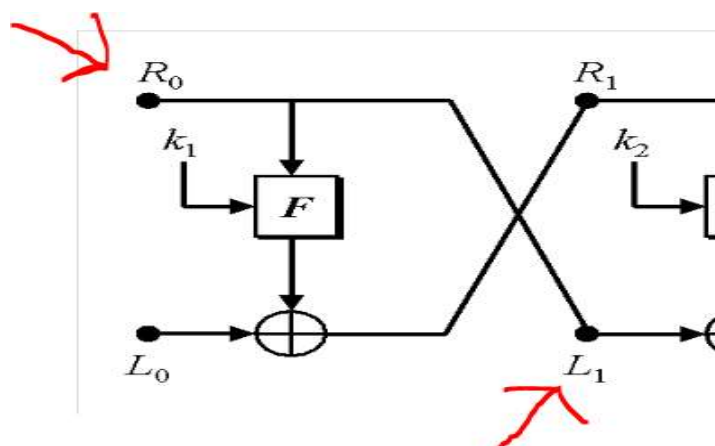
cipherCreditCardNumber += (L[ROUNDS] + R[ROUNDS])
```

Στην συνέχεια δημιουργώ μια **for** επανάληψη στην οποία προσομοιώνω το Feistel δίκτυο δηλαδή θέλω να επαναληφθεί η διαδικασία 4 φορές σύμφωνα με τους γύρους μου που δήλωσα παραπάνω.

for i in range (1, ROUNDS+1):

Στην συνέχεια ορίζω το L[i] να πάρει την τιμή του R[i-1].

Δηλαδή στον πρώτο γύρο το L[1]=R[0] όπως και στην εικόνα παρακάτω.



Στην συνέχεια ελέγχω σε ποιόν γύρο βρίσκομαι και ακολουθώ τα αντίστοιχα βήματα .

$L[i] = R[i - 1]$

if (i == 1):

Key = MyFirstKey

else:

Key = SubKeyGeneration(L[i], MyFirstKey, i)

Δηλαδή αν το $i=1$ τότε βρίσκομαι στον πρώτο γύρο οπότε θα πάρω το πρώτο κλειδί μου το αρχικό.

Στην συνέχεια όταν ξανά μπει στην επανάληψη δηλαδή έχει ολοκληρωθεί ο πρώτος κύκλος μας και έχει ξεκινήσει ο δεύτερος κύκλος τότε γίνεται ο έλεγχος και μπαίνει στο else .

Key = SubKeyGeneration(L[i], MyFirstKey, i)

Εδώ καλώ μια συνάρτηση που έχω ορίσει παραπάνω η οποία δημιουργεί τα υποκλειδιά μου.

```
def SubKeyGeneration(s1, s2, i):  
    #print ("GENERATING KEY #" + str(i))  
    #print ("S1: " + s1)  
    #print ("S2: " + s2)  
    SubKey = hashlib.sha256(s1 + s2).hexdigest()  
    #print ("SubKey: " + SubKey)  
    return SubKey
```

Την έχω ορίσει ως SubKeyGeneration(s1,s2,i):

Η οποία λαμβάνει τα ορίσματα για την δημιουργία του υποκλειδίου μας L[i],MyFirstKey,i

Και για να το δημιουργήσουμε χρησιμοποιούμε την συνάρτηση **hashlib.sha256(s1+s2).hexdigest()** αυτή είναι μια έτοιμη συνάρτηση την οποία καλέσαμε στην αρχή με την εντολή **import** τις παίρνει έτοιμες από της βιβλιοθήκες της python.

```
import sys, getopt, hashlib  
from math import exp, expm1
```

Αυτό που κάνει αυτή η συνάρτηση είναι να παίρνει τις δυο τιμές μας και δημιουργεί ένα υποκλειδί και το επιστρέφω στο αρχικό μου πρόγραμμα .

Η ίδια διαδικασία θα πραγματοποιηθεί και για τους υπόλοιπους γύρους μας. Δηλαδή θα δημιουργηθούν τα υποκλειδιά μας.

```
R[i] = xor(L[i - 1], My_Function(R[i - 1], i, Key))|
cipherCreditCardNumber += (L[ROUNDS] + R[ROUNDS])
```

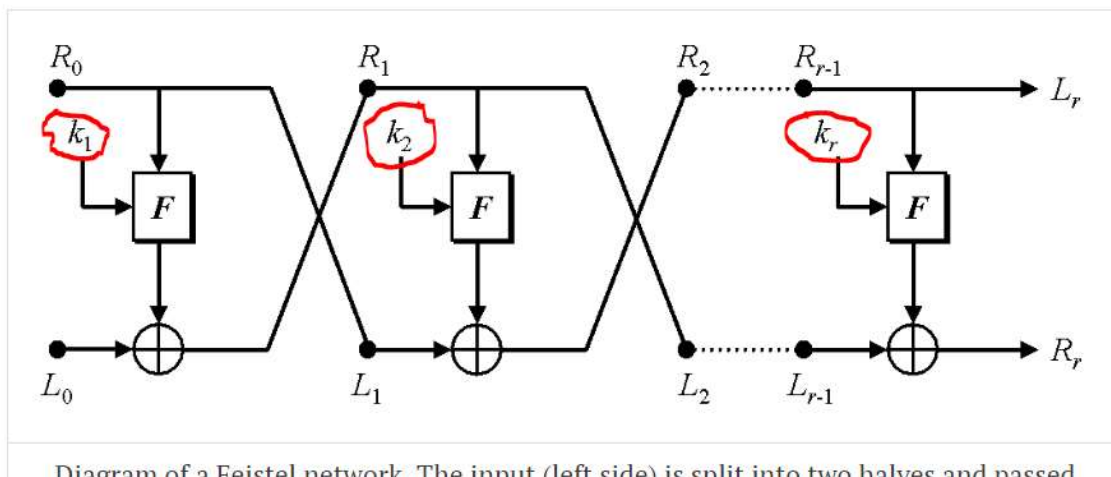
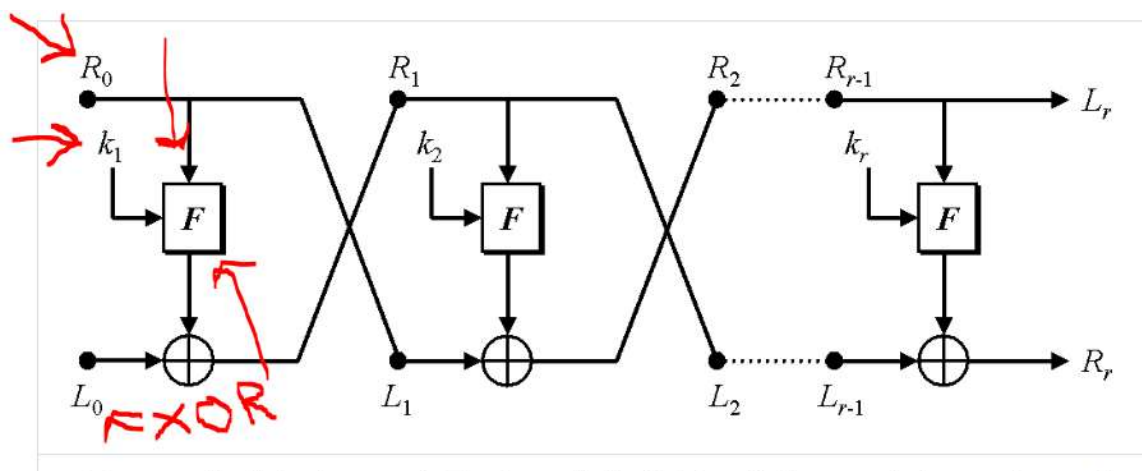


Diagram of a Feistel network. The input (left side) is split into two halves and passed

Τα κυκλωμένα είναι τα κλειδιά που δημιουργώ στους 4 γύρους μου.

Στην συνέχεια κάνω xor τις δυο τιμές ,περνάω μέσα από την συνάρτηση μου τις τιμή R[i] τον γύρο και το κλειδί που έχω δημιουργήσει και παίρνω σε κάθε γύρο το αντίστοιχο κλειδί .


$R[i] = \text{xor}(L[i - 1], \text{My_Function}(R[i - 1], i, \text{Key}))$



Οπότε κάνω xor το L0 με το αποτέλεσμα που προκύπτει από την συνάρτηση F().

Η συναρτήσεις που καλούμε είναι οι εξής .

My_Function όταν περνάμε το κλειδί και το $R[i-1]$ μέσα από την συνάρτησή **F ()**.



```
def My_Function(x, i, k):  
    k = ConvertStringToBinary(k)  
    x = ConvertStringToBinary(str(x))  
  
    k = ConvertBinaryToInt(k)  
    x = ConvertBinaryToInt(x)  
  
    res = pow((x * k), i)  
    res = IntegerToBinary(res)  
  
    return BinaryToString(res)
```

Δημιουργούμε την συνάρτηση μας η οποία λαμβάνει τις τρεις τιμές τον δυαδικό μέρος το κλειδί και τον γύρο.

```
def My_Function (x, i, k):
```

```
    k = ConvertStringToBinary(k) //καλώ την συνάρτηση μου και μετατρέπω το αλφαριθμητικό μου στο  
    διάδικο
```

```
# string to binary  
def ConvertStringToBinary(s):  
    return ''.join('{:08b}'.format(ord(c)) for c in s)
```

```
    x = ConvertStringToBinary(str(x)) //καλώ την συνάρτηση μου ConvertStringToBinary και μετατρέπω το  
    αλφαριθμητικό μου στο διάδικο
```

```
    k = ConvertBinaryToInt(k) //καλώ την συνάρτηση μου ConvertBinaryToInt και μετατρέπω από το  
    διάδικο στο ακέραιο.
```

```
    x = ConvertBinaryToInt(x)
```

```
# int to binary  
def IntegerToBinary(i):  
    return bin(i)
```

`res = row ((x * k), i)` Στην συνέχεια καλώ την συνάρτηση `row` η μέθοδος αυτή στην δύναμη.

```
import sys, getopt, hashlib
from math import exp, expm1
```

`res = IntegerToBinary(res)` //καλώ την συνάρτηση μου `ConvertBinaryToInt` και μετατρέπω από το διάδικό στο ακέραιο.

`return BinaryToString(res)` //Τέλος επιστρέφω σε δυαδική μορφή το αλφαριθμητικό.

Το αποτέλεσμα που έχω είναι το `R[i]`

```
R[i] = xor(L[i - 1], My_Function(R[i - 1], i, Key))
CipherCardNumber += (L[ROUNDS] + R[ROUNDS])
```

Όταν τελειώσει η επανάληψη μου δηλαδή έχουν ολοκληρωθεί όλοι η γύροι μου αναθέτω το αποτέλεσμα μου στην μεταβλητή **`cipherCreditCardNumber`** . Αυτό το αποτέλεσμα θα είναι το κρυπτογραφημένο μήνυμα.

```
print ( " Initial: " + K[0])
for i in range(1, ROUNDS+1):
    L[i] = R[i - 1]
    if (i == 1):
        Key = MyFirstKey
    else:
        Key = SubKeyGeneration(L[i], MyFirstKey, i)
    R[i] = xor(L[i - 1], My_Function(R[i - 1], i, Key))
cipherCreditCardNumber += (L[ROUNDS] + R[ROUNDS])
```

Στην ουσία παίρνει όλο τον πίνακα `L[ROUNDS]` και `R[ROUNDS]` και τα αθροίζει όλα μαζί.

`cipherCreditCardNumber += (L[ROUNDS] + R[ROUNDS])`

Τέλος τυπώνουμε το κρυπτογραφημένο κείμενο και το μήκος του κρυπτογραφημένου κείμενου στην κονσόλα μας.

```
print(cipherCreditCardNumber)
```

```
print(len(cipherCreditCardNumber))
```

Για το τρίτο βήμα της εκφώνησης σε περίπτωση που το μήκος του κρυπτογραφημένου κειμένου σε ακεραία μορφή είναι πολύ μεγάλο τότε παίρνουμε το κρυπτογραφημένο κείμενο και επαναλαμβάνουμε την διαδικασία .

(Επαναλαμβάνω το τρίτο κριτήριο.

3.Αλλά τι γίνεται αν το κρυπτογραφημένο κείμενο **C** είναι πολύ μεγάλο ? Κανένα πρόβλημα .Αν διαδικασία δεν ολοκληρωθεί με επιτυχία με την πρώτη τότε ξανά επαναλαμβάνουμε την διαδικασία μας. Παίρνουμε το Κρυπτογραφημένο κείμενο που προέκυψε **C** ,και το ξανά κρυπτογραφούμε για να προκύψει ένα νέο **C** πηγαίνοντας στο **Βήμα 2.**)

Τρέχω το πρόγραμμα μου:

Πηγαίνω στο command line και τρέχω το script μου .

Με την εντολή python **ErgasiaKruptografiap14086.py**

```
kwestas@kwestas-VirtualBox:~/Desktop$ python ErgasiaKruptografiap14086.py  
CreditCardNumber:4532294877918448  
10000000110100001100100100010001101110011010011110000  
L Initial: 1000000011010000110010010001001001000  
R Initial: 10001101110011010011110000  
CipherCreditCardNumber:  
<@@@wluuuuuuu)chq^@@8@@@[FQX]@@wuuuu^  
LengthOfCipherText:  
54
```

CreditCardNumber:4532294877918448

```
100000000110100001100100100010001101110011010011110000
```

```
L Initial: 100000001101000011001001000
R Initial: 10001101110011010011110000
```

```
CipherCreditCardNumber: CreditCardNumber))
<??itw??u????)chq??^??8????[?F?QX?)??w????^
```

Αν αυτός ακέραιος μπορεί να αποτυπωθεί τότε έχει ολοκληρωθεί η διαδικασία μας αν όχι επαναλαμβάνουμε την διαδικασία μας .

Προσέγγιση της μεθόδου με έτοιμες βιβλιοθήκες AES .

Θα μπορούσαμε να χρησιμοποιήσουμε στην μέθοδο που περνάμε μέσα από την συνάρτηση **F()** δηλαδή αντί για την συνάρτηση `my_Function()` δημιουργήσαμε στον κώδικα μας .Αρχικά θα κάναμε `import` όλες τις βιβλιοθήκες της κρυπτογράφησης της python

```
from Crypto.Cipher import *
```

From Crypto.Cipher import *

Στην συνέχεια θα παίρναμε το δεκαεξάρικό κλειδί που έχουμε και στο δικό μας πρόγραμμα θα δημιουργούσαμε και ένα `iv` που χρειάζεται ο Aes μας.

```
iv=''.join([chr(random.randint(0,0xFF))for i in range(16)])
```

Και θα χρησιμοποιούσαμε την παρακάτω εντολή για να το κρυπτογραφήσουμε χρησιμοποιώντας το κλειδί , την CBC mode , και το `iv` initialization vector .

```
aes= AES.new(key, AES.MODE_CBC, iv)
```

Πηγές Που Χρησιμοποιήθηκαν

<https://github.com/fgutica/COMP7401-Feistel-Cipher>

<https://www.youtube.com/watch?v=UB2VX4vNUa0>

Ολόκληρος ο Κώδικας:

```
File Edit View Search Tools Documents Help
Open + [F] Save

import sys, getopt, hashlib
from math import exp, expm1

import random

SECRET = "s2rj6235-77d3-3405-3d23-462w34179227"

def mykey(key):
    return hashlib.sha256(key + SECRET).hexdigest()

# string to binary
def ConvertStringToBinary(s):
    return ''.join('{:08b}'.format(ord(c)) for c in s)

# binary to int
def ConvertBinaryToInt(s):
    return int(s, 2)

# int to binary
def IntegerToBinary(i):
    return bin(i)

# binary to string
def BinaryToString(b):
    n = int(b, 2)
    return ''.join(chr(int(b[l: l + 8], 2)) for l in xrange(0, len(b), 8))

def My_Function(x, t, k):
    k = ConvertStringToBinary(k)
    x = ConvertStringToBinary(str(x))

    k = ConvertBinaryToInt(k)
    x = ConvertBinaryToInt(x)

    res = pow((x * k), t)
    res = IntegerToBinary(res)

    return BinaryToString(res)

# xor two strings
def xor(s1, s2):
    return ''.join(chr(ord(a) ^ ord(b)) for a, b in zip(s1, s2))

def SubkeyGeneration(s1, s2, l):
    #print ("GENERATING KEY #" + str(l))
    #print ("s1: " + s1)
    #print ("s2: " + s2)
    Subkey = hashlib.sha256(s1 + s2).hexdigest()
```



```

return ''.join(chr(ord(a) ^ ord(b)) for a, b in zip(s1, s2))

def SubKeyGeneration(s1, s2, l):
    #print ("GENERATING KEY #" + str(l))
    #print ("s1: " + s1)
    #print ("s2: " + s2)
    SubKey = hashlib.sha256(s1 + s2).hexdigest()
    #print ("SubKey: " + SubKey)
    return SubKey

CreditCardNumber=4532294877918448 #Its an example of a credit card number that the article give us.
print ("CreditCardNumber:" + str(CreditCardNumber))

keys = ''.join(chr(random.randint(0,0xFF))for i in range(16)) #create a random 16 byte key.

BinaryFormOfCreditCardNumber=bin(CreditCardNumber)[2:] #Calculate the binary form of the credit card number.

print (BinaryFormOfCreditCardNumber)
#print(len(BinaryFormOfCreditCardNumber))

ROUNDS = 4
BLOCKSIZE = 54
BLOCKSIZE_BITS = 54

cipherCreditCardNumber = ""
n = BLOCKSIZE
BinaryFormOfCreditCardNumber = [BinaryFormOfCreditCardNumber[i:i+n] for i in range(0, len(BinaryFormOfCreditCardNumber), n)]
#print(BinaryFormOfCreditCardNumber)

lengthOfLastBlock = len(BinaryFormOfCreditCardNumber)-1

if ( lengthOfLastBlock < BLOCKSIZE):
    for i in range(lengthOfLastBlock, BLOCKSIZE):
        BinaryFormOfCreditCardNumber[len(BinaryFormOfCreditCardNumber)-1] += " "

TemporaryKey = mykey(key)
MyFirstKey = TemporaryKey

for block in BinaryFormOfCreditCardNumber:
    #print ("Block: " + block)
    L = [""] * (ROUNDS + 1)
    R = [""] * (ROUNDS + 1)
    L[0] = block[0:BLOCKSIZE/2]
    R[0] = block[BLOCKSIZE/2:BLOCKSIZE]
    print ("L Initial: " + L[0])
    print ("R Initial: " + R[0])
    for i in range(1, ROUNDS+1):
        L[i] = R[i - 1]
        if (i == 1):

```

```

BinaryFormOfCreditCardNumber[len(BinaryFormOfCreditCardNumber)-1] += " "

print (BinaryFormOfCreditCardNumber)
#print(len(BinaryFormOfCreditCardNumber))

ROUNDS = 4
BLOCKSIZE = 54
BLOCKSIZE_BITS = 54

cipherCreditCardNumber = ""
n = BLOCKSIZE
BinaryFormOfCreditCardNumber = [BinaryFormOfCreditCardNumber[i:i+n] for i in range(0, len(BinaryFormOfCreditCardNumber), n)]
#print(BinaryFormOfCreditCardNumber)

lengthOfLastBlock = len(BinaryFormOfCreditCardNumber)-1

if ( lengthOfLastBlock < BLOCKSIZE):
    for i in range(lengthOfLastBlock, BLOCKSIZE):
        BinaryFormOfCreditCardNumber[len(BinaryFormOfCreditCardNumber)-1] += " "

TemporaryKey = mykey(key)
MyFirstKey = TemporaryKey

for block in BinaryFormOfCreditCardNumber:
    #print ("Block: " + block)
    L = [""] * (ROUNDS + 1)
    R = [""] * (ROUNDS + 1)
    L[0] = block[0:BLOCKSIZE/2]
    R[0] = block[BLOCKSIZE/2:BLOCKSIZE]
    print ("L Initial: " + L[0])
    print ("R Initial: " + R[0])
    for i in range(1, ROUNDS+1):
        L[i] = R[i - 1]
        if (i == 1):
            Key = MyFirstKey
        else:
            Key = SubKeyGeneration(L[i], MyFirstKey, i)

        R[i] = xor(L[i - 1], My_Function(R[i - 1], i, Key))

    cipherCreditCardNumber += (L[ROUNDS] + R[ROUNDS])

print ("CipherCreditCardNumber:")
print(cipherCreditCardNumber)
print ("LengthOfCipherText:")
print(len(cipherCreditCardNumber))

```


Κώδικας για Χρήση ΑΕΣ δεν υπάρχει στο κανονικό μου αρχείο .

```
Card2.py (~/Desktop) - gedit

a=4532294877918448
bin(a)

print (bin(a)[2:])
print len(bin(a)[2:])

import random

key= ''.join(chr(random.randint(0,0xFF))for i in range(16))

print ("key",[x for x in key])

iv=''.join([chr(random.randint(0,0xFF))for i in range(16)])

from Crypto.Cipher import *

aes= AES.new(key, AES.MODE_CBC, iv)

data ="hello world 1234" #<-16bytes

encd= aes.encrypt(data)

print (encd)

aes=AES.new(key, AES.MODE_CBC, iv)
decdec=aes.decrypt(encd)

print (decdec)
```